**Question 1**

**Web Application Development**

**Implement a web application using Java, Spring boot and Oracle SQL to efficiently read 1 million records from an oracle table**

1. **Created an Oracle table INFOSYS\_EMPLOYEES with the below columns**

CREATE TABLE INFOSYS\_EMPLOYEES (

EMPLOYEE\_ID NUMBER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,

FIRST\_NAME VARCHAR2(50) NOT NULL,

LAST\_NAME VARCHAR2(50) NOT NULL,

EMP\_EMAIL VARCHAR2(100) UNIQUE NOT NULL,

PHONE\_NUMBER VARCHAR2(20),

HIRE\_DATE DATE DEFAULT SYSDATE,

JOB\_TITLE VARCHAR2(50),

SALARY NUMBER(10,2),

DEPARTMENT\_ID NUMBER

);

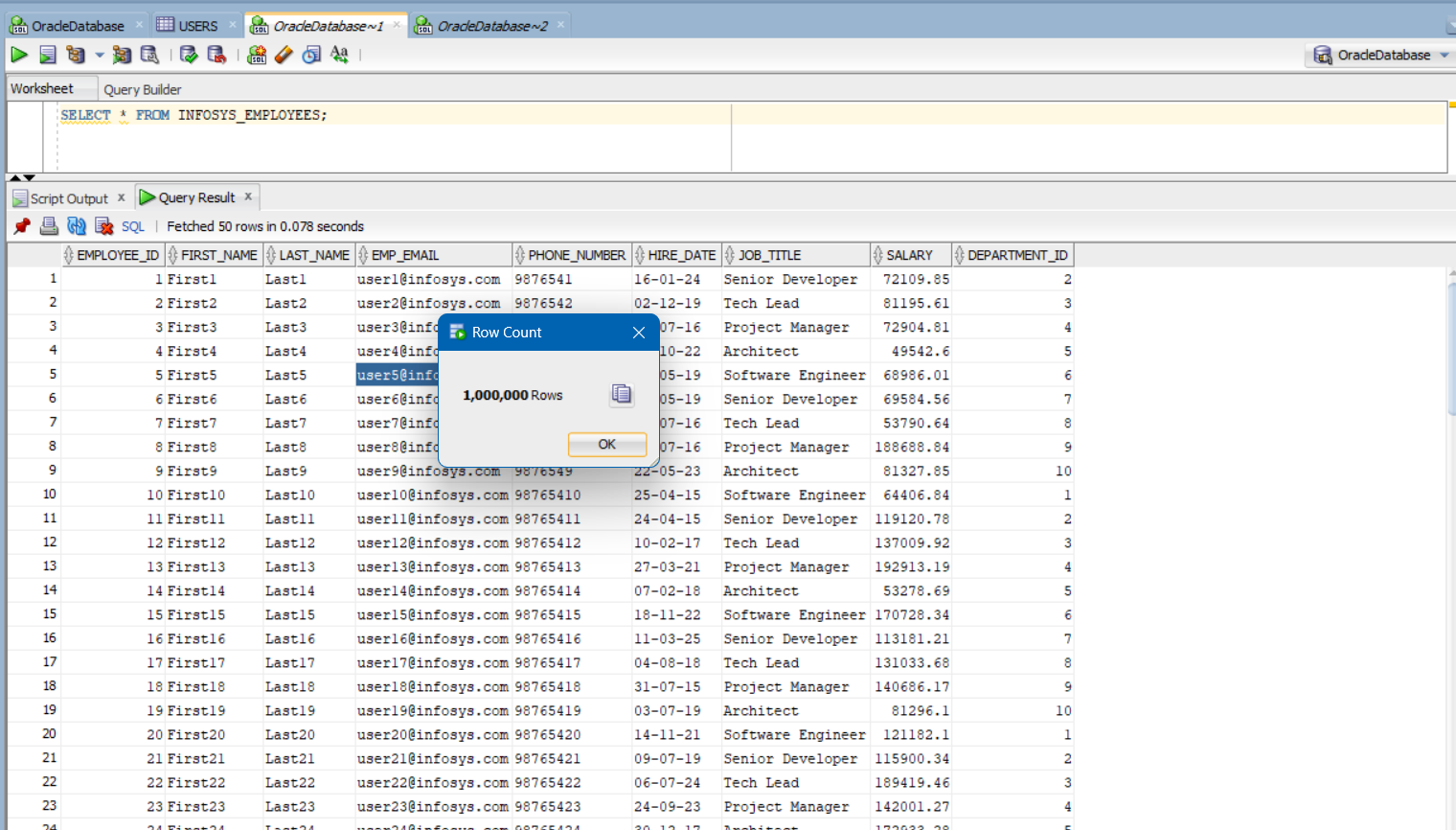
1. **Created Index for faster query processing**

CREATE INDEX INDEX\_EMP\_EMAIL ON INFOSYS\_EMPLOYEES(EMP\_EMAIL);

CREATE INDEX INDEX\_EMP\_JOB\_TITLE ON INFOSYS\_EMPLOYEES(JOB\_TITLE);

CREATE INDEX INDEX\_EMP\_DEPT ON INFOSYS\_EMPLOYEES(DEPARTMENT\_ID);

1. **Inserted 1 million entries in the employee table**



**Approaches we can adopt while working with millions of records**

1. Instead of fetching all the records at once and cause high memory usage and slow performance it is better than we implement pagination and only retrieve data in smaller chunks using LIMIT or ROWNUM
2. Make use of Indexing on the database side which helps speeds up query execution. If our query searches without an index, it performs a full table scan, which is slow. Hence we can add indexes on columns used in WHERE or ORDER BY clauses.
3. Fetching all the columns from the database wastes memory and bandwidth. Instead, fetch only the required fields using a DTO (Data Transfer Object) to reduce unnecessary data load.
4. We can make use of Asynchronous Processing so If fetching takes time, it’s better to fetch in the background. We can make use of @Async in Spring Boot.
5. Make use of Caching to avoid fetching result of repeated Queries. f you're repeatedly fetching the same data, it’s better to store it in a cache instead of hitting the database.

@Cacheable("employees")

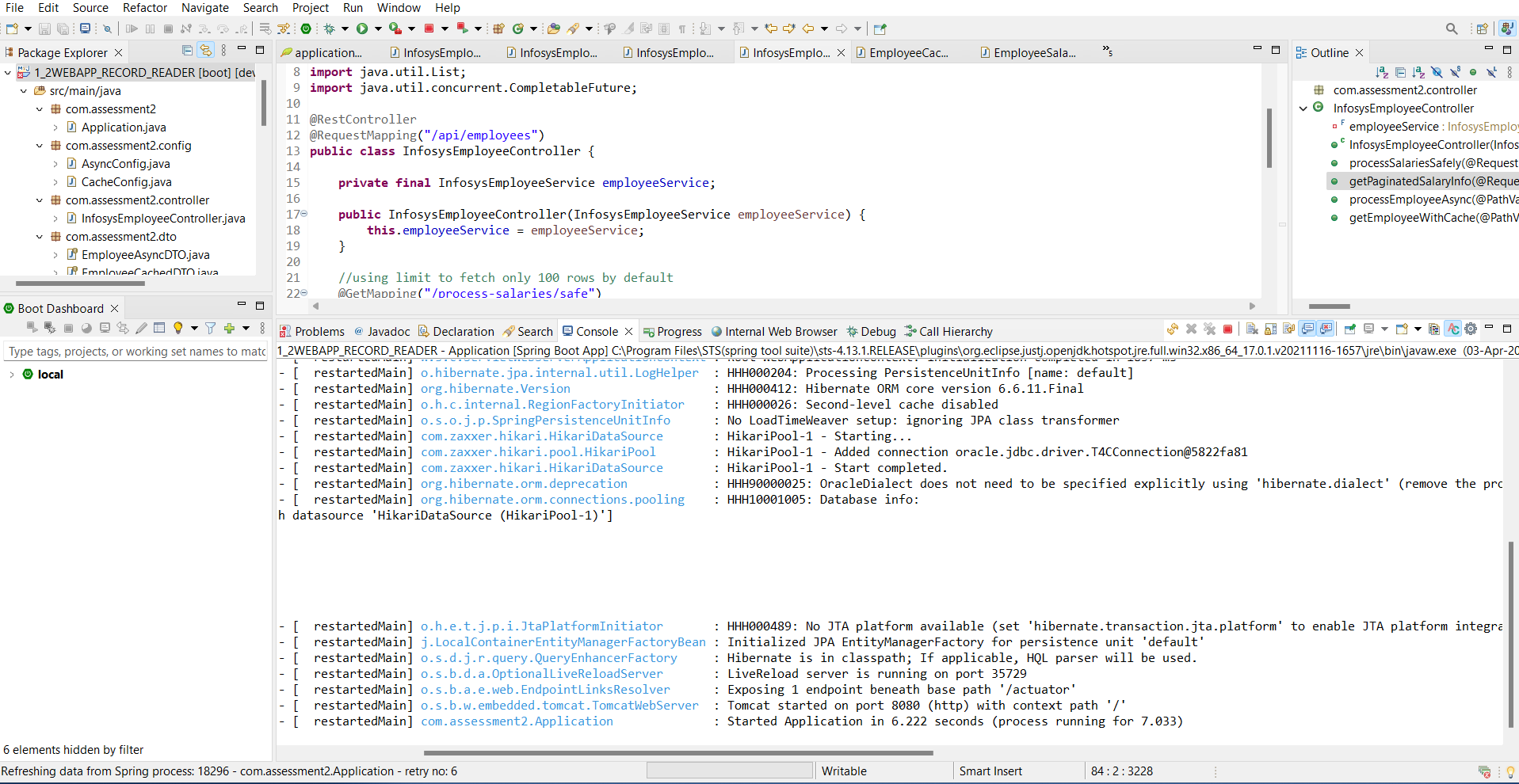
public List<Employee> getEmployees() {

return employeeRepository.findAll();

}

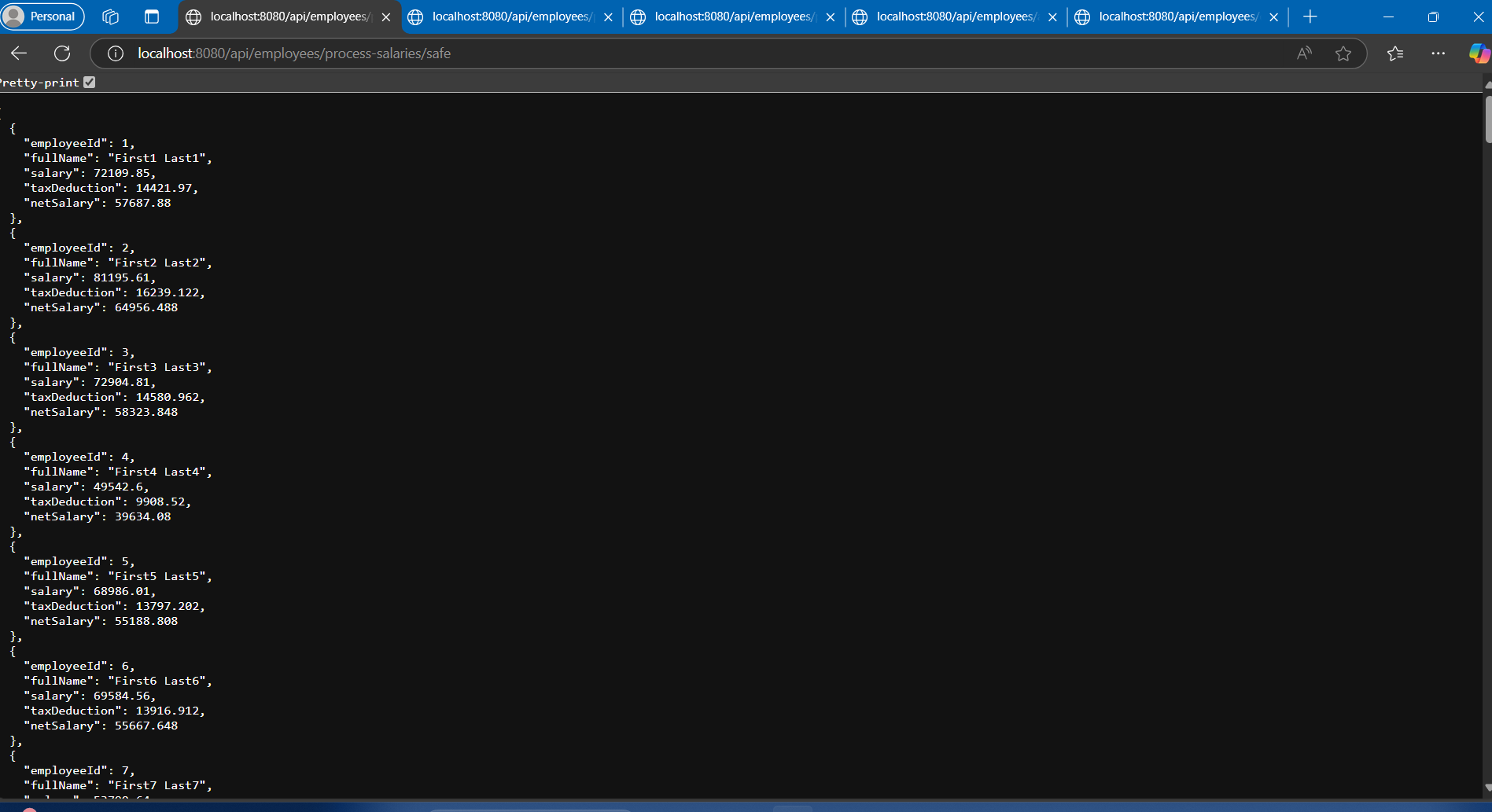
1. We can make use of Caffeine Caching which is a In-Memory Cache for Faster Reads, it provides super-fast caching inside the same JVM and hence better to use this if in our application runs on a single server. Also, if your application doesn’t require to be persisted (data is lost if the app restarts).
2. We can make use of Redis if we have multiple instances of your application (microservices). We need to share caching between different servers. And it allows persistent caching (Redis stores data even after app restarts).

Application is up and running

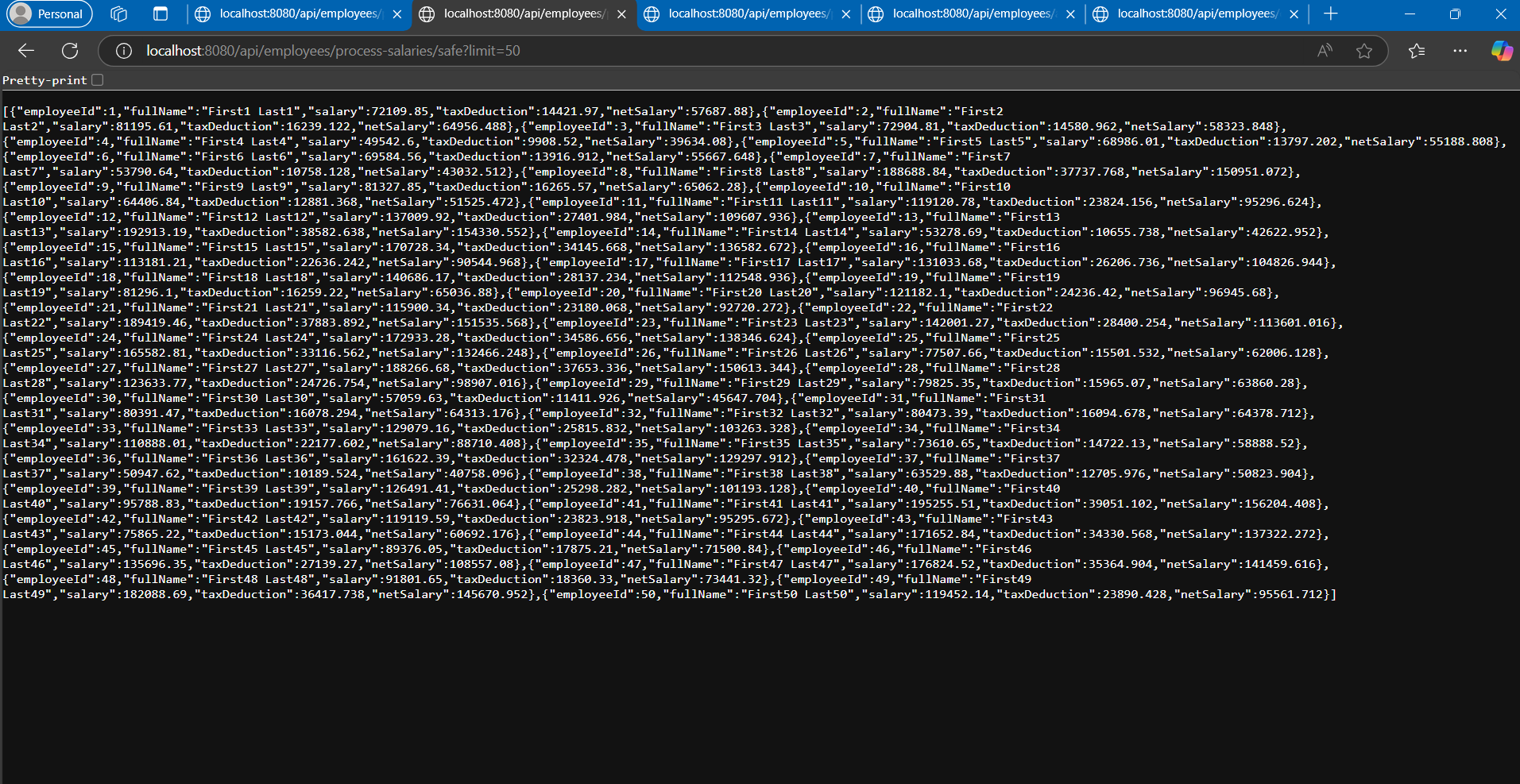


**Testing the Endpoints and result evidences**

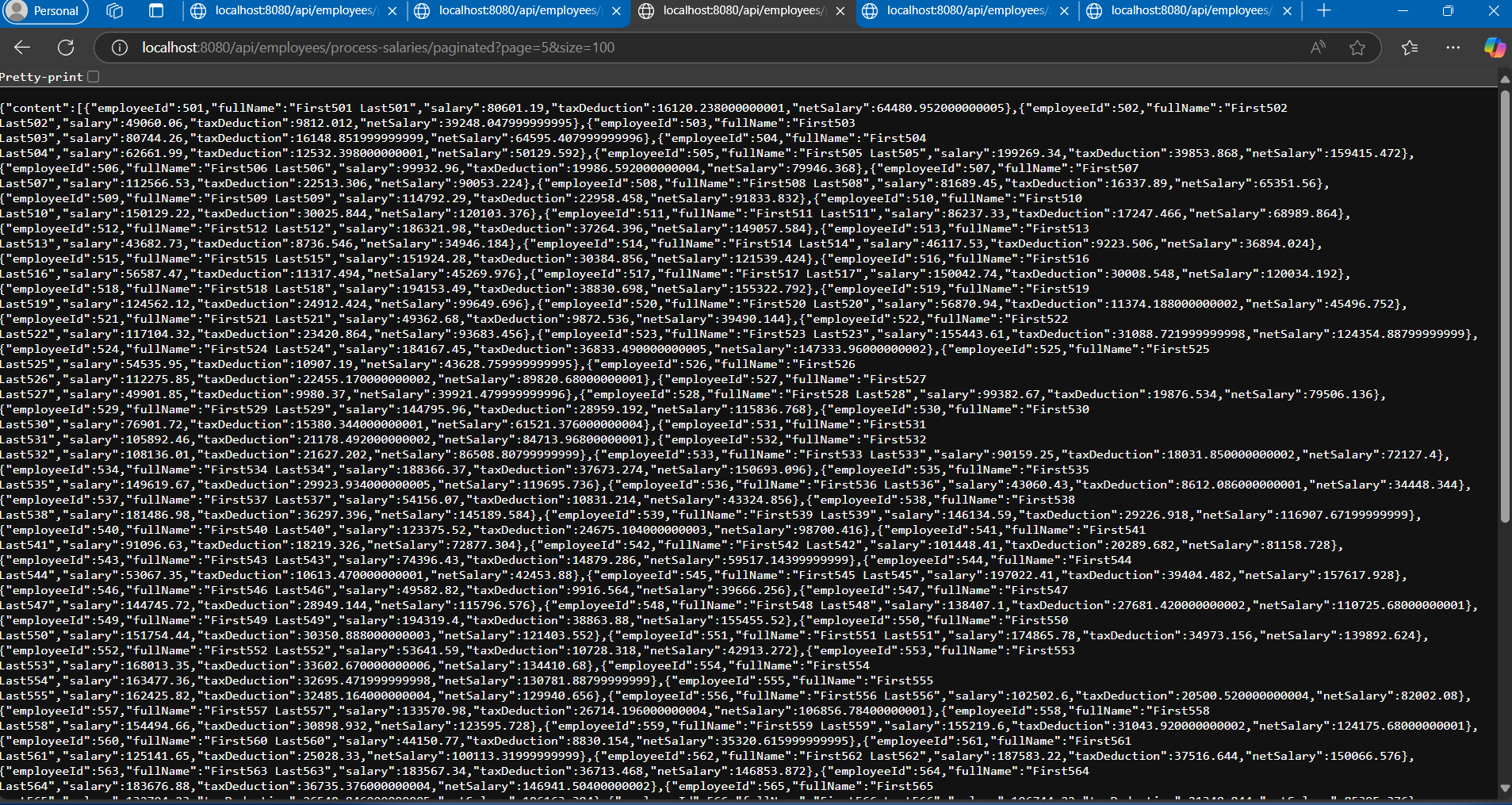
[**localhost:8080/api/employees/process-salaries/safe**](http://localhost:8080/api/employees/process-salaries/safe)

****

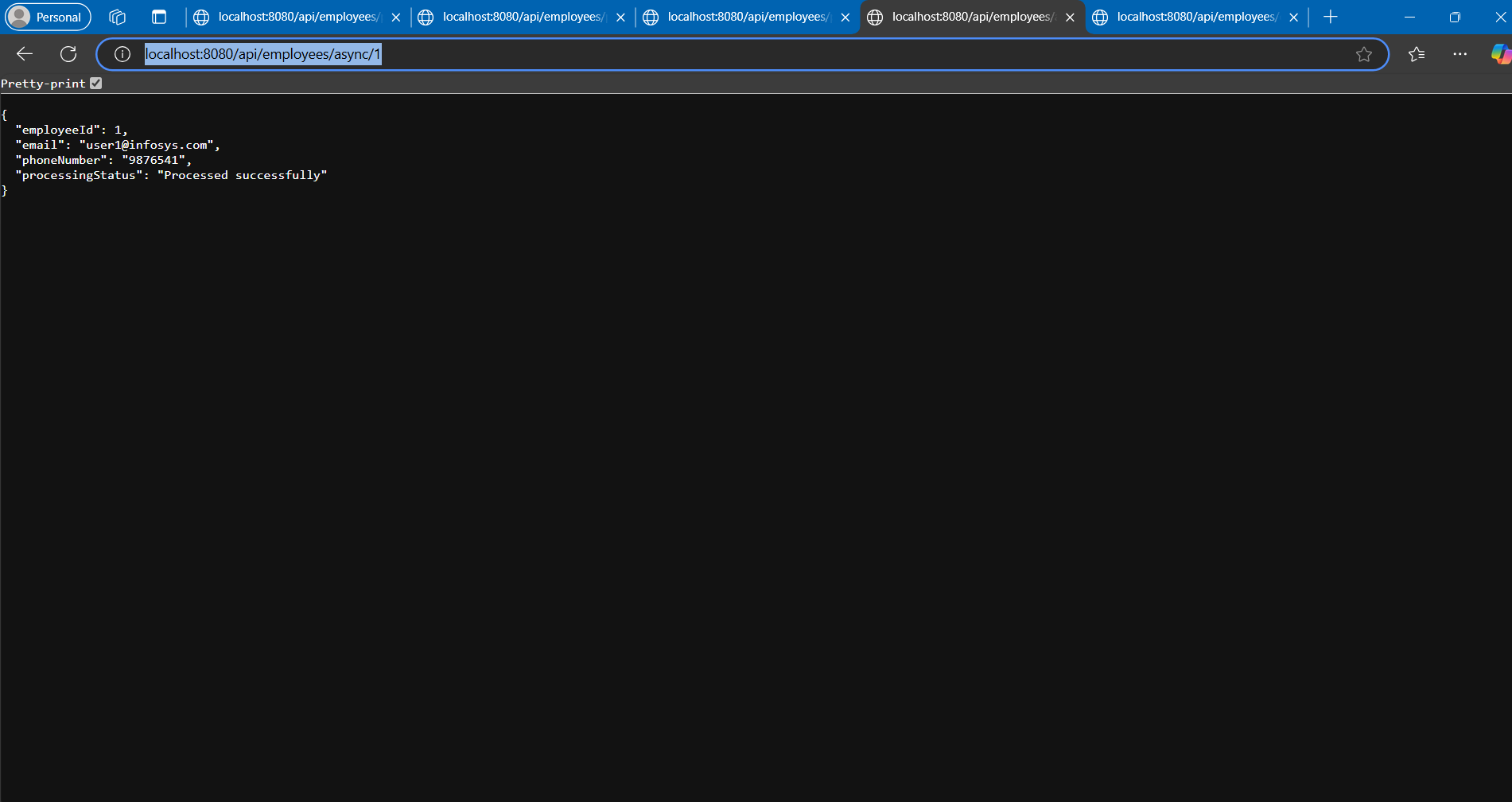
[**localhost:8080/api/employees/process-salaries/safe?limit=50**](http://localhost:8080/api/employees/process-salaries/safe?limit=50)

****

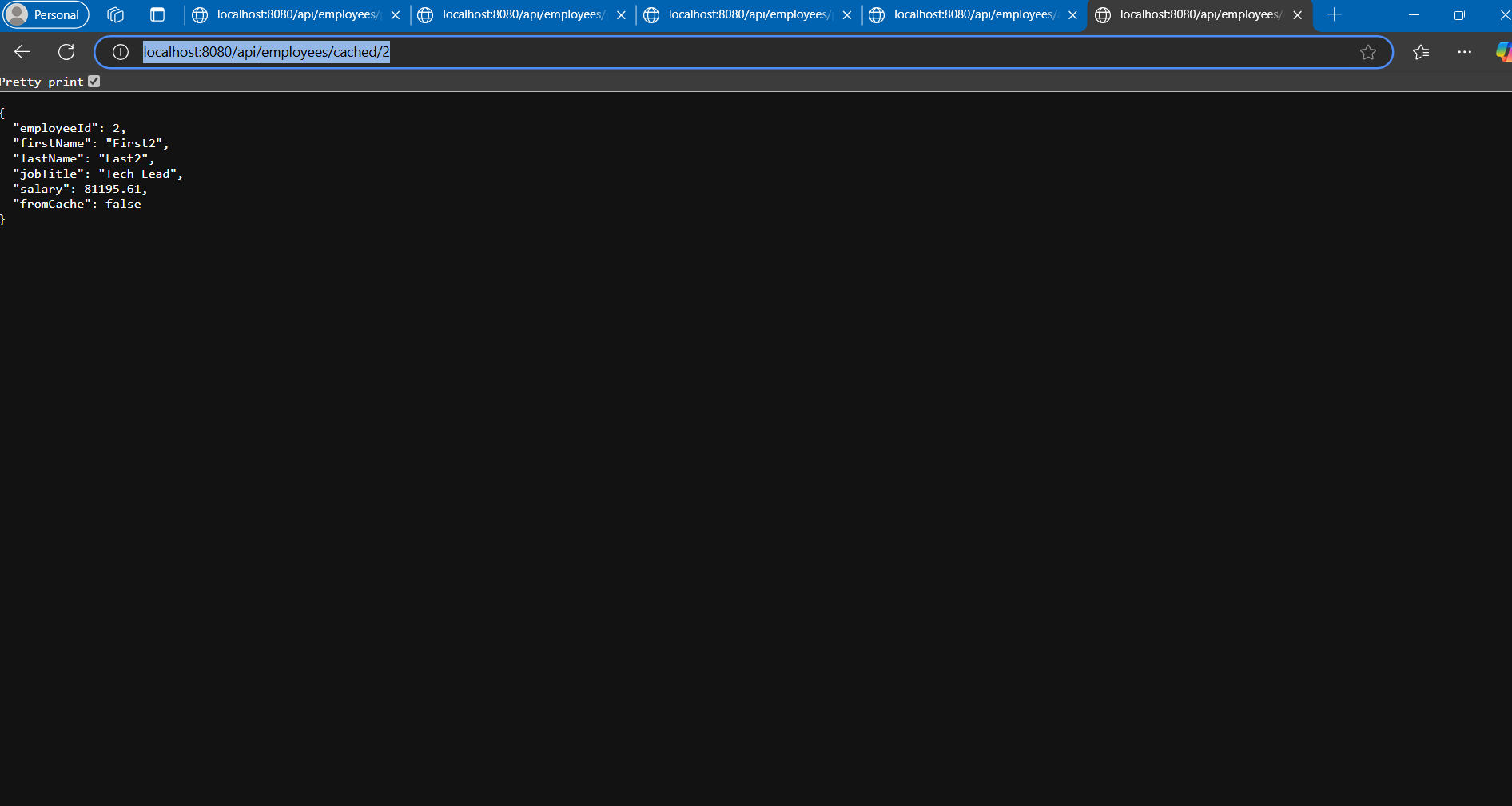
[**localhost:8080/api/employees/process-salaries/paginated?page=5&size=100**](http://localhost:8080/api/employees/process-salaries/paginated?page=5&size=100)

****

[**localhost:8080/api/employees/async/1**](http://localhost:8080/api/employees/async/1)

****

[**localhost:8080/api/employees/cached/2**](http://localhost:8080/api/employees/cached/2)

****